

Cluster File systems for LHC

Rainer Többsicke, CERN/IT

Draft 1 - 20031106

Usage

We consider cluster file systems in three different usage scenarios:

1. Home directory, software distribution style: files typically small (hundreds of bytes to dozens of megabytes), high access rate (10000 file accesses total, mostly trivial (non-data-transfer)), relatively small in total (10s of TB) with more than 99% availability. Good locality of reference, making caching efficient. Transfer speed to individual files below 10 MB/s, about 1 GB/s global transfer capacity. Most of the space is to be backed up to tape nightly.
2. Analysis: random access, 1000/s total frequency, little locality of reference making disk/memory cache relatively useless. Transfer speed around 100MB/s for individual files, 50 GB/s total. Total space requirement 100s of TB.
3. Central data recording: mostly sequential access, 100 interactions/s (with the staging system), total space requirements in the order of PBs, aggregate transfer speed 5 GB/s, data mainly resides on magnetic tape.

Features of a cluster file system and related issues

Transparent access vs. library access, semantics

Files are either accessed in the standard operating-system defined way, using the traditional operating system notation (e.g. path names) and interfaces (e.g. C-library), or through an abstraction library isolating from the underlying file system implementation (the latter being only practical for a limited set of applications as it requires the application to collaborate). The file system supports traditional access styles (read, write, read+write, append) and implements standard semantics (seek, truncate) for sequential and random access.

Security

- Cluster-only access, authentication is performed by the cluster, needs methods to identify which nodes make up the cluster. Access from non-cluster nodes (untrusted machines) through gateways. All server elements of the file system (file servers, disks) have to be shielded from other nodes.
- Integrated into an authentication framework, e.g. Kerberos. Clients nodes need not be trusted.

Scalability

How does the system scale with

- the number of files,
- the number of simultaneously open files,
- the number of hosts accessing,
- the number of disks in the system,
- the size of the disks.

SAN-model versus NAS model

In the traditional NAS (network attached storage) implementation of a shared file system data reside on disks local to the file server. The file server access data on behalf of the client and passes them over the network. In this model the fileserver also performs basic space allocation and acts as a synchronization point for simultaneous data access from multiple network nodes. Typically it also implements authorisation checking and access control.

In storage area networks storage devices and hosts are interconnected and disks directly accessible from multiple hosts at the same time. This does not automatically mean data can be freely shared, though: structural information (e.g. a list of unallocated disk blocks, uniqueness of directory entries, file locks) has to be maintained by all participating hosts in a synchronized manner. In the old 'mainframe' days this was sometimes implemented using a 'lock' or 'reserve' on the disk itself. Today's SAN file system implementations usually introduce a 'lock manager' or 'metadata server'. Clients address requests to access data to a metadata server as before, using the normal (e.g. IP) network. The data itself however take the direct path over the storage area network. At the cost of an additional step, this technique of separating the 'control' and the 'data' path brings several advantages:

1. SAN protocols are light weight and low latency compared to a full IP network. Access to data is typically as fast e.g. SCSI.
2. The 'metadata server' is smaller than the file server in the NAS world, more readily tunable for the small transactions that make up the load required to maintain control over how data is accessed.
3. In its most basic configuration data is accessed directly through the disk controller, eliminating the bottleneck represented by the file server in NAS configurations where the server has more than one disk.
4. SANs would be almost useless without it.

Object disk storage

In this model 'disks' (read: controllers with possibly several spindles) are directly attached to the SAN. However, some of the file system intelligence has migrated into the disk controller, approaching again a traditional NAS model. Object storage devices store 'objects' – bytes strings that can be part of files and already have certain attributes (e.g.

Access rights, time stamps, etc.) - and perform basic space allocation. The 'security' problem attached to plain SAN disks is meant to be resolved here as the 'object disk' will perform some sort of authentication. There is a SNIA standard (www.snia.org/tech_activities/workgroups/osd/) describing object disks.

Networking

TCP/IP

Infiniband

RDMA

Myrinet

Management:

File system space management means work. A sort of Hubble constant of 1 FTE/TB is currently believed to be the de-facto price to do it 'by hand' which is out of question. Therefore, some **automatic, policy-based** management is needed:

1. Selection of disk(s) when files are being created;
2. quota per major unit (experiment, project);
3. delegate management of part of the storage system;
4. non-disruptive reconfiguration, on-line migration of files for hardware changes, backup while system is running.

Scaling

Number of expected clients: 5000; number of expected disks: 5000

Data migration

Ideally the file system should interface to an HSM system such as CASTOR.

The Candidates

AFS

AFS has the reputation of being slow for big file access mainly because of inflexible local disk caching and the underlying RX protocol. AFS does not support more than around 64k files in a single directory. AFS does not support files bigger than 2GB. AFS does not support full POSIX semantics for accessing files.

But:

AFS is well tuned for access to 'small' files. AFS is secure. AFS is suited for World-wide access. AFS has a flexible access control mechanism. AFS space management is largely automated. AFS is Open Source. AFS is scalable. AFS is very well established in many high energy physics sites.

AFS has been around for more than 15 years and is since 3 years Open Source. A group of 'AFS-Elders' comprised mainly of people from US Universities manage changes to the code brought in by the AFS user community. The communities' emphasis is currently stability and porting to new platforms (e.g. MacOS X, FreeBSD) with a strong bias for Linux (different distributions, 64-bit Linux on AMD & Intel). There is moderate ongoing effort on new features such as Kerberos 5 support, Large File Support, improved Windows support. Only few suggestions depart from the original design of AFS which is now over 15 years old, although this is changing.

The following areas in particular interest in the view of using AFS at LHC startup:

1. Large file support: standard AFS currently only supports files smaller than 2 GB. Support for large files recently went into the fileserver-side.
2. SAN-based storage, Object disk storage: there have been tests at Carnegie Mellon University with *Network Attached Secure Disks* – something like a precursor of today's Object Disk Storage.
3. Links to SAN file systems: in this scenario the AFS file server stores files in a SAN file system (such as GPFS, GFS, Storage Tank). Files are served in the usual way to AFS clients without access to the SAN, clients on the SAN however are re-directed to using the underlying SAN file system directly.

Performance

Traditional AFS performance is hampered by the local disk cache (segments are stored before given to application), the RX protocol (measured 2/3 of TCP performance) on big transfers, the transfer size which is the same for all files in a given node, and lastly the file server performance. Tests bypassing the AFS cache shows that the latter is actually no problem.

'Extended' AFS using the underlying cluster file system (GPFS) above performs at about the speed of the cluster file system: in this case AFS is operated as a metadata server performing access control and managing the file system topology, something which does not induce a significant overhead.

Currently under test at CERN: AFS on Ethernet-connected 'Object Disks'. A special flag in an AFS file instructs the cache manager to transfer the requested blocks from a disk server over a TCP connection. Standard AFS files are metadata only, files do not contain data but the address(es) of the 'object disk(s)'. As in the SAN file system case, the process has several advantages at the cost of an extra indirection:

1. Data is transferred using TCP instead of RX. TCP is well tuned and a single stream can almost saturate a gigabit Ethernet connection.
2. The AFS cache is bypassed: data is transferred directly into the application buffer instead of going through the virtual memory system, avoiding 1 extra copy and (in the case of big buffers) latency delays due to fragmentation of requests.
3. The extra indirection breaks the rule 'all files of a single AFS directory on the same disk (or LUN)' of traditional AFS. Files can be allocated on a set of disk servers in round-robin or any other (e.g. 'overflow') fashion. The design also allows for mirroring and to a certain extent striping.

Scaling

AFS has few “hard” limitations with respect to scaling (the most annoying one being the number of files per directory). We don't foresee any issues with the number of clients for the LHC era nor with the number of servers (so there is no immediate limit on the total amount of data which could be stored in AFS).

Practically however:

- In standard AFS a directory always resides on a single “disk”. The extension proposed weakens that limit.
- In standard AFS a single directory with large amounts of data (100s of GB) becomes increasingly 'unmanageable' with standard AFS tools. Everything works, but the management of such large chunks will be cumbersome. Again, the proposed extensions change the characteristics of the problem by placing it on a per file level.

Storage Tank

Storage Tank is implemented as a cluster of Metadata servers and attached SAN (Fibre channel, iSCSI) disks. The Metadata servers run a SAN-based database that contains file information (names, attributes, disk block addresses) and the list of free disk blocks. The Storage Tank (SANFS) clients requests a data descriptor from the server and directly accesses the disk. Based on a 'lease' technique Storage Tank guarantees full local file system semantics, even with simultaneous readers/writers on separate nodes. Storage Tank also aims at full heterogeneity (Windows, Linux, AIX, Solaris,...), by maintaining full local file system semantics.

Storage Tank does not implement any RAID function (other than what the underlying disk controller offers).

Security

In its current incarnation Storage Tank does not implement any security worth mentioning. File access is controlled by uid/gid matching on the file system client (as the protocol uses unprivileged ports, this is however even less secure than NFS) . Any node on the SAN nevertheless has complete control over all disks. This positions the file system as a cluster file system.

Scaling

Each disk (or LUN) is a separate device accessible by all Storage Tank clients. For the iSCSI configuration tested here most Linux systems have a default limit of 42 such devices, a limit which can be set in the kernel to 128. This is less than half of what we have installed as disk servers today, let alone disks. Tests are under way to further check out this limit, but it is unclear whether this scales by orders of magnitude without any serious penalty!

Management

One of the main selling arguments for Storage Tank is **policy-based lifecycle data management**. Today this is limited to a filename-based disk selection rules on file

creation, which have already been criticised for the lack of flexibility.

Storage Tank in its first release also lacks an interface to a data migration system. This is planned for future releases.

Lustre

Basic design similar to Storage Tank: metadata is stored in a cluster of tens of servers with transaction-based synchronization.

OST (object storage target)

Data is stored on object storage targets, from Linux servers to eventually individual disks. (standards?) Allocation and free space management is performed by the object itself. A file consists of a set of objects.

OST are accessed using an access method called 'portals' developed at Sandia lab, implementing a light-weight ('0-copy on send') message passing protocol. The method isolates the file system from the underlying network infrastructure which can IP, Infiniband, Myrinet, etc.

The metadata cluster uses a distributed lock manager from IBM. Access control is implemented using NIS-style userid/groupid authorisation.

Security

All nodes fall into the same security domain and trust each other for access rights.

InterMezzo

Started at CMU by Peter Braam. Focussed on reliability, emphasis on data replication, disconnected operation. Log-based file system, whole file caching.

Latest project news fall 2002.

Panassas

Object disk based file system. Brand new. Founded by someone from CMU parallel data laboratory.

Others

File system current not investigated deeply but deserving further interest:

- NFS (V4)
- DAFS

Strategy

The two areas 'software distribution' and 'CDR' are covered using today's AFS and CASTOR technology. AFS is well-established in the HEP world and has received a boost in stability since the community took up fixing bugs when AFS turned Open Source,

making it plausible that it will 'survive' throughout the running of LHC. IT will have to continue investing in CASTOR development at some level unless a viable alternative solution is currently not in sight.

The area 'analysis' is not fully covered as of today, however it is likely that a mixture of AFS and CASTOR will be able to fill the hole.

In parallel, we continue to investigate alternative technologies:

- Extensions to traditional AFS as described above in two flavours (object disk based/SAN fs based). While retaining full functionality (security/world-wide access, policy-based management, single name space, etc.) of the current AFS infrastructure this would bring the performance required to fully cover the 'analysis' area, while at the same time offering a standardized replacement for the CASTOR 'disk server' layer.
- Storage Tank: ongoing Openlab project with IBM Storage Tank is interesting in particular since it supports iSCSI, whereas similar designs require a true SAN.
- Lustre: we should test this system as it looks like a serious alternative for Storage Tank.